Basics of Computing – Chapter 5 Algorithms

Cory L. Strope

University of Nebraska Lincoln

Algorithms

Developing Algorithms

Recursion

Motivation
Problem Solving

We have talked about multiple problem solving methodologies:

- Two's complement $\leftrightharpoons$ binary
- Gate construction using AND, OR, NOT
- Timesharing / Multitasking
- Fetch – Decode – Execute
- Token Ring / Bus protocols

Motivation
Problem Solving

Sorting Problem:

- Given a list of numbers:

12	18	5	24	2

- We want to sort the list:

2	5	12	18	24

Problem Solving
G. Polya (1945)

Given a problem:

- Four phases of problem solving:
  1. Understand the problem
  2. Devise plan for solving the problem
  3. Carry out the plan
  4. Evaluate the solution for accuracy
- Problem solving does *not* have to be sequential.

Problem Solving

Computer Science

Given a problem:

- Develop an approach for solving the problem:
  - Understand the problem – What are the preconditions? Postconditions?
  - Devise a solution – Preconditions $\rightarrow \ldots \rightarrow$ postconditions.
  - Express the solution so that *even a computer* can understand.
  - Check solution for correctness.
- Problem Example: Make Toast!

Problem Solving
Toast

1. Acquire bread, toaster and plate.
2. Place 1 piece of bread in toaster.
3. Push lever down.
4. Wait until toaster finishes.
5. Pick up bread
6. Place bread on plate.
7. Repeat until enough toast is made.

Problem Solving

Toast

1. Acquire bread, toaster and plate.
2. Place 1 piece of bread in toaster.
3. Push lever down.
4. Wait until toaster finishes.
5. Pick up bread
6. Place bread on plate.
7. Repeat until enough toast is made.

Potential flaws:

1. What if bread is moldy? How do we handle the situation?
2. Is the toast done well enough? What setting should the toaster be on?
3. Is the plate large enough? Is the hole for the toaster large enough?
4. How many pieces of bread do we have? How much toast do we want?

Algorithm

Definition

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

Informal Definition: A collection of steps that does a specific task.

Algorithms

Developing Algorithms

Recursion

Methods

There are various methods for representing algorithms:

- A computer program – Algorithms understandable by a machine.

Algorithms are abstract – They represent *concepts*.
How do we create *physical representations* of concepts?

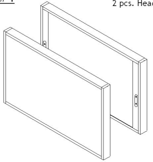
- Programs
- A sequence of pictures
- Flow chart
- Pseudocode

Bed Assembly Instructions BARBADOS BED

Parts And Fittings Packing Detail

Box # 1

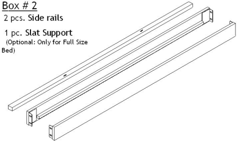
2 pcs. Head Board



Box # 2

2 pcs. Side rails

1 pc. Slat Support
(Optional: Only for Full Size Bed)

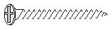


Hardware Pack In Box #1

JCBC Bolts
M8 x 30mm - 8 pcs

Washer - 8 pcs.

M4 x 32mm Screw
- 26 pcs.



M5 Allen Key 1 pc.

Optional Hardware For Full Size Bed Only

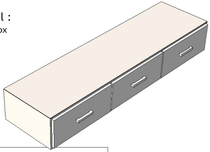
M4x 16 mm Screw
2 pcs

Slat Support Bracket
2 pcs

M4x 20 mm Screw
4 pcs

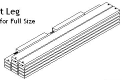
Optional :

Drawer Box



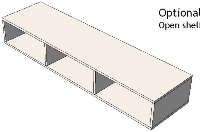
Box # 3

13 pcs.Slats
2 pcs. Support Leg
(Optional: Only for Full Size Bed)

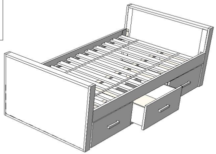


Optional :

Open shelf

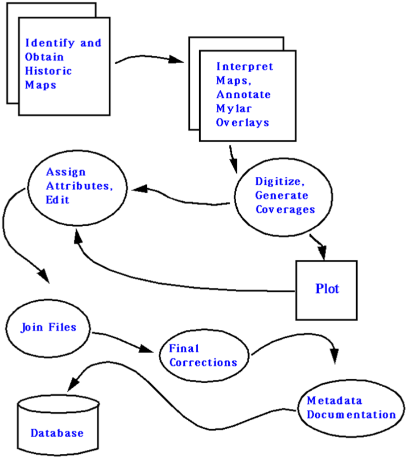


A complete assembled view of the Barbados Bed



LIFESTYLE SOLUTIONS
The Fusion Of Function And Comfort With Style

Flow Charts


Identify and Obtain Historic Maps
Interpret Maps, Annotate Mylar Overlays
Digitize, Generate Coverages
Assign Attributes, Edit
Plot
Join Files
Final Corrections
Metadata Documentation
Database

Pseudocode

Pseudocode is an outline of a program; an informal representation of the algorithm with common language.

- Enables you (the "programmer") to concentrate on the algorithm.
- Has a structure and syntax that is similar to many modern programming languages.
- Can be easily converted to a program.
- Is easily understandable by a human (*not a machine language*).

Pseudocode will be the preferred method of writing algorithms in this course.

I will write pseudocode using `this font`.

Pseudocode

Comments and Assignment

Comments are *non-executable* steps of the algorithm, but provide the programmer with some background information.

- Syntax: `// Some comments`
- Everything after the `//` on a line is considered to be a comment.
- Comments can take an entire line, or simply come after a valid instructions.

The **assignment operator** allows us to give a value to some variable.

- `VARIABLE` ← `VALUE`
- For example:
  - `y` ← `32`

Pseudocode

There are 3 common models to use in algorithm development:

1. Sequential
2. Decision Making or Selection
3. Repetition

Sequential

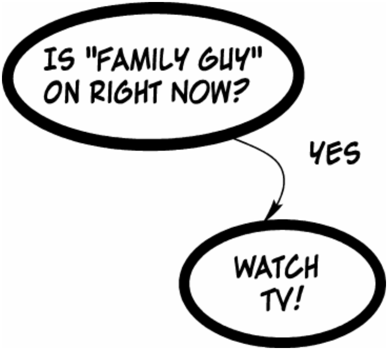
Processes happen one after another, no decisions or repetitions are necessary.

Syntax:

```
Step 1
Step 2
etc.
```

Selection Methods

if-then



There is a single process, either do or not do.

```
if(condition) then {
    // Perform action
}
```

condition has *only 2* evaluations: true or false.

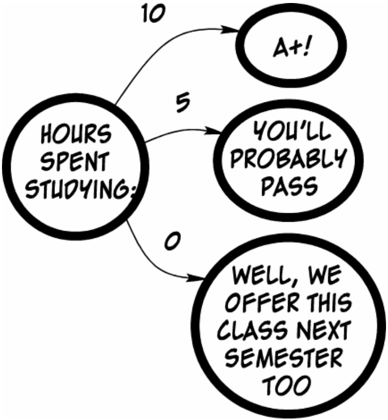
Note: Operations that are related to the `if` statement are indented.

Choice Methods
if-then-else

Choice between two processes.

```
if(x = 1) then {
    x ← y
} else {
    x ← x + 1
}
```

Choice Methods
switch



Multiple potential options exist, choose one.

```
if(hours ≥ 10) then {
    grade ← A
} else if(hours ≥ 5) then {

    grade ← B
} else {
    // No comment...
}
```

Repetition Methods

Repetition methods, or **loops**, allow us to execute a set of steps multiple times. There are two main types of Repetition methods:

- Conditional Loops
  - Continuously executes a set of steps while some condition is true.
- Iterative Loops
  - Executes a set of steps a predefined number of times.

Repetition Methods
while

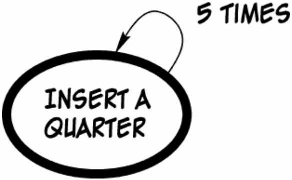
Conditional loop: Repeat an actions a (unknown) number of times.

```
while(condition) do {

    // steps
}
```

When the condition fails, we continue with the next sequential instruction *after* the loop. Similar to JUMP.

Repetition Methods
for



Iterative loop: You need to repeat an action a (known) number of times.

```
for VARIABLE ← BEGIN to END {
    // steps
}
```

`VARIABLE` starts at `BEGIN`, and

- increments (`for i ← 1 to n`) or
- decrements (`for i ← 10 to 1`)

by 1 every iteration until it reaches `END`.

Repetition Methods
foreach

Related to the `for` loops is the `foreach` loop:

```
foreach ELEMENT in STRUCTURE {
    // do steps
}
```

A STRUCTURE can be a list of elements. For example:

foreach x_i in X {
    // do steps
}

where X is a list, and x_i is an element in the list.

Pseudocode
Primitive Variable Types

A **primitive** is a well-defined set of building blocks from which algorithm representations can be constructed.

- Primitives consist of two parts:
  1. **Syntax**: Symbolic representation
  2. **Semantics**: Meaning of the primitive
- Each of the previous methods is a primitive:
  - `for`, `while`, `if-then`, `if-then-else`, ...

The *variables* (`VARIABLE`) are ambiguous:

- `y` $\leftarrow$ `32`
- x_i, X (i.e. `foreach` x_i `in` X)

How?

Pseudocode
Primitive Variable Types

Recall chapter 1:

People understand a large number of symbols:

{a–z, A–z, 0–9, &, %, #, ...}

{a, aardvark, ..., zulu, zygote}

Pictures

Sounds

Type and size *of* **text** written

i.e., e.g., et al, etc; etc; etc.

Computers do these processes using their symbol library:

{0, 1}

0s and 1s are ambiguous! They can be (for pseudocode):

- Numbers:
  - Floating-point (`double`)
  - Two's complement (`int`)
  - *Less commonly:* Unsigned binary
- ASCII codes (`char`)
- Arrays or Lists of `double`, `int`, `char` (X).

Pseudocode
Other Statements

Other statements are necessary for a proper pseudocode language:

- Printing: `print "Hello"`
- Complex Assignment: `x` $\leftarrow$ `(y` $\times$ `z) / 2`
- Return: `return X` from a function (next subsection).

If other operations are necessary, the statements need to be descriptive, clear, and concise.

Pseudocode
Common Tasks

Searching an array for a number, call it `num`:

X =	x[0]	x[1]	x[2]	x[3]

```
for i ← 0 to 3 do {
    if(X[i] = num) {
        print "found number"
    }
}
```

Pseudocode

Function Header

Algorithms expressed in pseudocode needs to have a header, in the form:

```
ALGORITHM_NAME(inputs)
```

- `ALGORITHM_NAME` should describe what the algorithm does.
- `inputs` are the **arguments** to the algorithm.

For example, given the previous list of numbers $\{x_1, x_2, \ldots, x_n\} \in X$

```
SORT(X) {
    // the steps of the sort algorithm.
}
```

Pseudocode
Function Calling

The function header and associated pseudocode is called a **function** or **procedure**.

- Functions are subprograms that accomplish a specific task
- Functions can be called within other functions by writing down the *name* of the procedure and supplying the *inputs*.

Pseudocode
Function Calling

Given $\{x_0, x_1, \ldots, x_{n-1}\} \in X$, A list with n elements:

```
FOO(X,n) {
    for i ← 0 to n-1 {

        if(x_i > x_{i+1}) {
            SWAP(X,i,i+1)
        }
    }
}
```

```
SWAP(X, i, j) {
    temp ← x_i
    x_i ← x_j
    x_j ← temp
}
```

Pseudocode

Function Return Values

```
// Print odd numbers
// between start and end.
PRINT_ODD(start, end) {
  for i ← start to end {
    if(IS_ODD(i)) then {
      print(i)
    }
  }
}
```

```
// If num is odd, return true
// otherwise, return false.
IS_ODD(num) {
  isOdd ← false
  if(num % 2 = 1) then {
    // % is modulus
    isOdd ← true
  }
  return isOdd
}
```

Pseudocode

Insertion Sort

To get an idea of how the insertion sort works, run this program on the list: X = {5, 3, 1, 9, 12, 4, 21, 18, 7, 9}.

```
INSERT_SORT(X,n)
   for i ← 0 to n-1 {
      iOfLarj ← FIND_KEY(X,i,n)
      SWAP(X,i,iOfLarj)
   }
}

SWAP(X,i,j) {
   temp ← X[i]
   X[i] ← X[j]
   X[j] ← temp
}
```

```
FIND_KEY(LIST,listStart,listEnd) {
   index ← LIST[listStart]
   for i ← listStart to listEnd {
      if(LIST[i] > index) {
         index ← LIST[i]
      }
   }
   return index
}
```

Motivation
Algorithms